LA-UR-01-0892

Approved for public release; distribution is unlimited.

Title: REPORT ON THE IMPACT OF PROPOSED ARCHITECTURAL CHANGES TO THE 30T ARCHITECTURE, INCLUDING THE INFLUENCE OF BANDWIDTH ON ASCI APPLICATIONS

Author(s):
Henry J. Alme, CCS-3
Richard F. Barrett, X-8
Marion Kei Davis, CCS-3
Daryl w. Grunau, CCS-1
Adolfy Hoisie, CCS-3
Fabrizio Petrini, CCS-3
Dean A. Prichard, CCS-1
Sunlung Suen, CCN-8
Harvey J. Wasserman, CCS-3

Submitted to: CCS-3 Techincal Report

Los Alamos
NATIONAL LABORATORY

Report on the impact of proposed architectural changes to the 30T architecture, including the influence of bandwidth on ASCI applications

Hank Alme, Richard Barrett, Kei Davis, Daryl Grunau, Adolfy Hoisie, Fabrizio Petrini, Dean Prichard, Sunlung Suen, Harvey Wasserman

Executive Summary

This memo is our response to your request that we determine the bandwidth requirements for important applications at LANL. The request was motivated by the possibility that the local, point-to-point bandwidth in a future network may be only 200 MB/s instead of 500 MB/s and by the decreased size of the SMP boxes.

We have shown that major apps from the ASCI workload, Rage, Partisn, an EOLUS project application, and a Classified Application, **are not bandwidth sensitive and that, given the overall design of the network, the decrease in bandwidth from 500 to 200 Mbytes/s will not lead to any significant performance degradation**. Moreover, the increase in bandwidth we would need to experience in order to achieve bandwidth saturation is high, 2 orders of magnitude roughly for Rage. This is an unlikely possibility due to the necessary increase in memory sizes, of roughly 3 orders of magnitude for Rage, that would be needed. The EOLUS project app has a different communication pattern, and while in principle this pattern can become bandwidth bound, we identify ways to alleviate this. This analysis is carried out in Sections 3 and 4.

In addition, our analysis indicates that the Q network is adequate at sustaining heavy traffic, as shown in sections 2 and 3.

Our bandwidth-related conclusions are based on input from actual runs on Blue Mountain and Nirvana, accurate simulation of the Q network and on models validated on all ASCI platforms.

We also touch on the issues of reliability (in section 5) and operating system overheads in section 6. We signal a potentially significant extra overhead from reducing the number of processors per SMP due to the fact that, since one processor per SMP handles system tasks, more processors will be dedicated to running OS.

A concern related to the increased complexity of system administration activities is presented in section 7.

We show that the static allocation of rails proposed by Compaq is not achievable because it would require an impractical number of rails. A dynamic allocation of rails will be

needed, but it will come at the price of available bandwidth degradation for bidirectional traffic, as that encountered in some of our important applications. We also analyze striping of large messages across rails, reaching the conclusion that due to a limitation in the design of the PCI bus, this bandwidth-enabling solution will not be feasible in MPI. The network analysis is presented in section 8.

Hoping to help the ASCI execs in exploring the technical issues and in their negotiations, we present below a more detailed analysis of each of these conclusions.

This analysis will continue beyond this report and this deadline, using a larger spectrum of data sizes, other classified apps and in general performance and scalability issues related to this architecture.

1. Introduction

This memo is our response to your request that we determine the bandwidth requirements for important applications at LANL. The request was motivated by the possibility that the local, point-to-point bandwidth in a future network may be only 200 MB/s instead of 500 MB/s.

In considering the bandwidth requirements for an application, it is necessary to understand bandwidth from two points of view. In addition to the local, point-to-point bandwidth available to a node we also have to understand the global, aggregate bandwidth available to an application running on all nodes in the network.

We gain this understanding by first considering some basic characteristics of the Q network. We then turn to a combination of experimental studies, simulation, and analysis.

2. Basic Network Considerations: Bisection Bandwidth of a Fat-Tree.

An important measure of aggregate bandwidth is "bisection bandwidth," which is the bandwidth across a cut in the network that separates the network into exactly two equal halves. Bisection bandwidth is an accurate measure when many nodes are communicating at the same time, so it helps us understand worst-case behavior of the network. As is shown below, bisection bandwidth is highly dependent on the topology of the network.

In a simple example let's assume a network connecting 4 processors in a square mesh, as depicted in Figure 1.

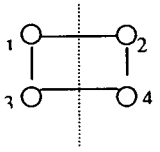

Figure 1. Bisection Bandwidth for a 2-D Mesh

Let's assume that the communication is composed of 2 point-to-point messages: processor 1 communicates to processor 2 and processor 3 to processor 4. The bisection bandwidth B_s in this example, as suggested by the dotted line is:

$B_s = 2*B_{ptp}$,

where B_{ptp} is the point-to-point bandwidth. It is clear in this example that the physical connectivity in the network allows the two messages to be sent concurrently, i.e. the time needed for the two messages to complete is the same as the time needed for a single message. The factor of two in the in the formula for bisection bandwidth indicates the availability of enough physical connectivity for the two messages to be handled in parallel.

The bisection bandwidth of a fat-tree, the topology of the 30T interconnect, is:

$B_s = N/2 * B_{ptp}$,

where N is the number of nodes attached to the network. For the 30T architecture N=384. This formula assumes 1 rail interconnecting each SMP to the network.

Conclusion: **Bisection bandwidth of fat-trees is high; in fact, it scales linearly with the number of nodes, insuring a high-level of message concurrency in these networks.**

3. Simulation Studies: Bandwidth Under Random Traffic

Besides communication library traffic generated by applications, the 30T network needs to handle all the I/O traffic. Under heavy I/O loads from many processors in addition to messaging from apps, it is conceivable that the network traffic becomes random. Let's assume a scenario in which one particular application runs on the 30T network. We assume that the subgrid allocation to processors can be done quasi-optimally so that pairs or groups of processors involved in communications are physically close in the network, for example belonging to the same switch or to adjacent switches. A favorable property of fat-trees is that in this case the network traffic will be "localized". In general this experiment is designed so that the traffic is minimized, a best case scenario. The higher levels of switches would likely see very little traffic.

A second scenario, that of random traffic, assumes that processors anywhere in the machine are involved in communications. In fact, we assumed that no communication involves processors that belong to the same switch, a worst case scenario.

Simulations were performed for the two cases using a high-resolution, instruction-level simulator of the network and of the network interface. The simulator was developed at LANL and validated by measurements on real networks. Bandwidth was calculated in both cases as the average of the bandwidth in all point-to-point communications, a metric

which we call "average random bandwidth" (ARB). We found minor differences in only, at the first decimal digit, in ARB in the two cases.

***Conclusion*: The 30T network can sustain heavy loads and unfavorable traffic patterns.**

4. Experimental Studies of ASCI Applications

In absence of instrumentation at hardware level on the network card, it is impossible to directly measure the bandwidth achieved or required by an application. Therefore, we devised experiments to indirectly gauge the sensitivity of codes to available network bandwidth. We used several important ASCI applications that are representative of the largest consumers of cycles on the Blue Mountain machine.

4.1 Rage

To gauge the sensitivity of Rage to bandwidth we "mimicked" a change in bandwidth by progressively increasing the size of MPI messages in the code by factors of 2, 4, 10, 100, 500 and 1000. While doing these changes we insured that the code produced correct results. We used 2 input decks, those utilized for all scalability tests of Rage on all ASCI architectures. One major difference between the two input decks is that one turns mesh refinement on. Figure 2 presents the timing results for these input files. The key result is that the runtime is practically constant for values of the bandwidth up to 100 times that of the real bandwidth requirements of the code in all 4 experiments presented.

In order to understand why there is this bandwidth insensitivity, we instrumented the code to give information on the number of messages and message size. We found that in each cycle of this run approximately 500 messages were sent, with sizes varying from 32 to approximately 8K bytes, for the input file with no mesh refinement, and approximately 1200 messages ranging from 32 to 32K bytes for the second input deck. Notably, the number of larger messages is very small - small messages dominate in number by far. The small sizes of these messages and the relatively small number of messages leads to low global bandwidth requirements.

***Conclusion*: The practical conclusion is that for these input files the code is insensitive to changes in bandwidth because MPI messages are small and there are few of them.**

One interesting observation is that for the largest message sizes, i.e. the highest consumers of bandwidth in this code, the actual value of the point-to-point bandwidth on the Quadrics network is the highest, as it can be seen in from Figure 3. Note that the saturation value of the bandwidth for MPI is achieved somewhere between 8 and 32 Kbytes! This has important practical implications as well. The above Rage runs used an input deck that specified hydro only and a problem size of about 13,000 zones per processor. It is reasonable to ask whether the results obtained from the bandwidth study would carry over to other input decks using other physical processes and/or other sizes.

We cannot comment on the former; however, we firmly believe that the results are relevant to any other problem size. This is because of surface-to-volume relationship of computations to communications. Even if the computation

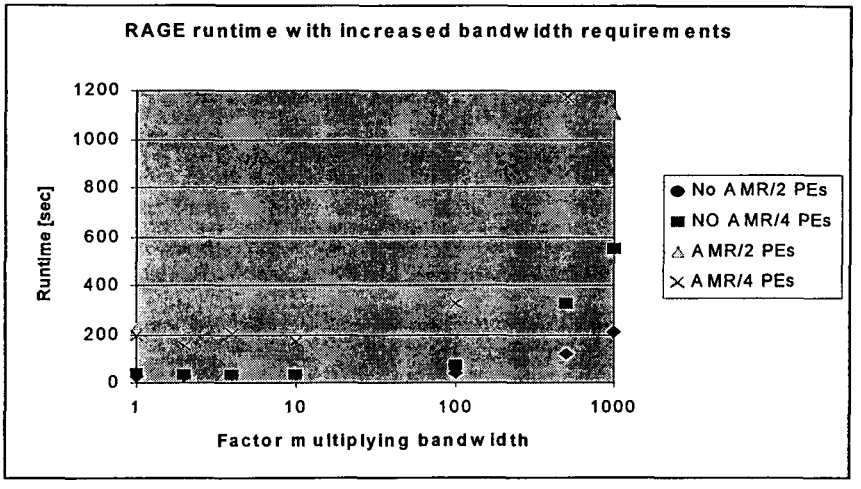

Figure 2. Bandwidth Sensitivity of Rage.

were scaled up, the communication (i.e., the message size) would only scale as the 2/3 power of the linear subgrid size. Available per-processor memory on any system we can conceive of would restrict the problem size. Hence, message sizes (the surface of the volume represented by the subgrid size per processor) would grow much more slowly. In order to reach the factor of 100 seen in Figure 2, at which point bandwidth would be a bottleneck, the problem size per processor would need to increase by a factor of 1000. This necessarily implies an increase of similar size in the memory available on each node.

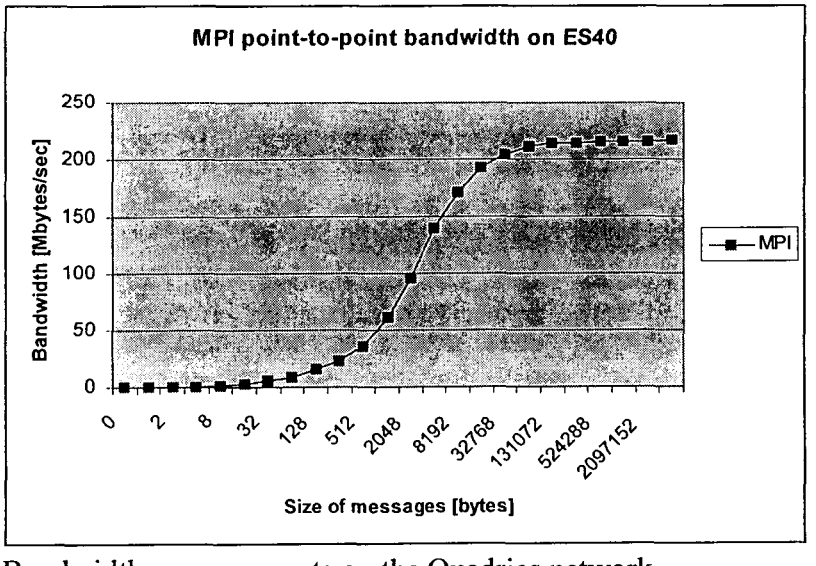

Figure 3. Bandwidth measurements on the Quadrics network

4.2 SWEEP / Partisn

For this experiment we used SWEEP3D, which completely and accurately mimics the communication pattern in Partisn. The differences between the two codes are in the computational aspects, not in the communication. For this analysis we used an analytical model of SWEEP3D developed at LANL, fully tested on all ASCI and other parallel architectures.

S_N transport experts indicated to us that the data sizes of interest for S_N transport calculations range from 5K to 16K cell points per subgrid. Figures 4 and 5 show a normalized runtime with increasing values of machine point-to-point bandwidth. Figure 4 is for the largest subgrid size, while Figure 5 is for the smallest.

For the bandwidth ranges of interest in this analysis, between 200-500 Mbytes/sec, the improvement in the communication time due to an increased bandwidth is small. The percentages on the plots represent the ratio between communication and computation in

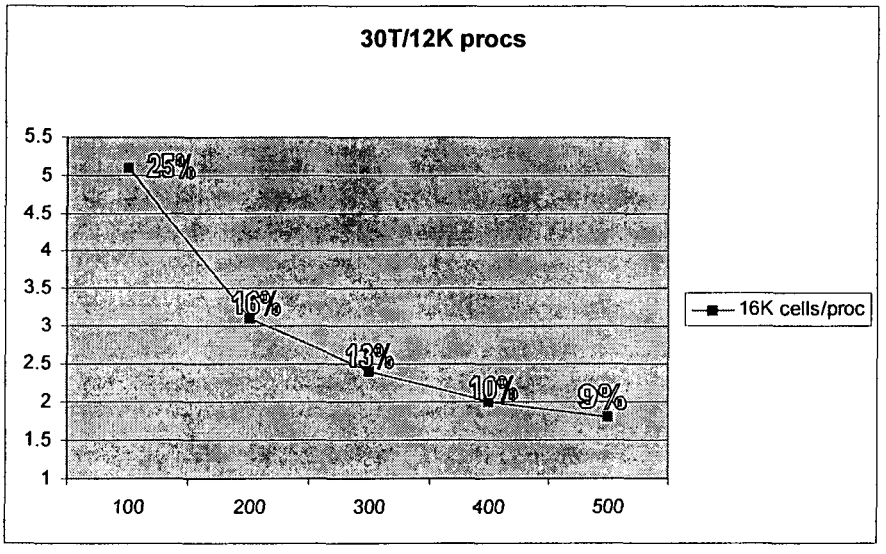

Figure 4. Runtime for SWEEP3D for 16K cells/proc

these runs. We see that this percentage varies only slightly. An improvement in bandwidth from 200 to 500 MB/s will only improve this ratio by approximately 6%. For comparison, for the small subgrid size, an improvement in latency by a factor of 2, from 5 to 2.5 μs, would lead to a 30% increase of the communication time.

***Conclusion*: The practical implication is that for the data sizes of interest, communication time is largely dominated by latency in S_N transport computations.**

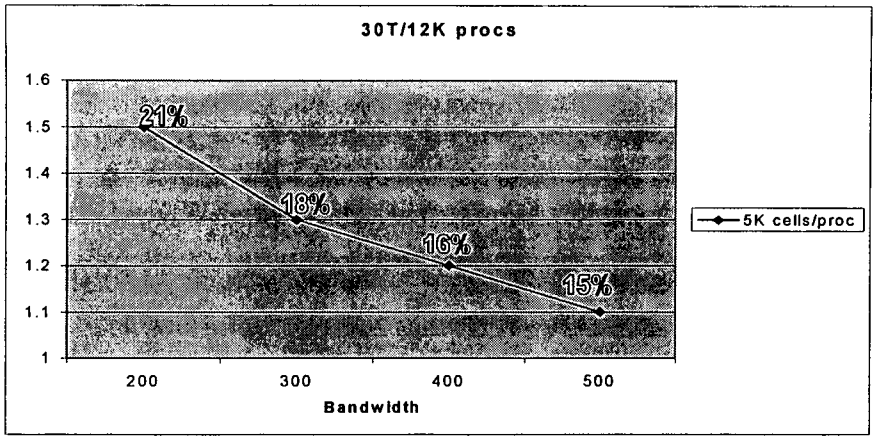

Figure 5. Runtime for SWEEP3D for 5K cells/proc

4.3 EOLUS Considerations

The main application in the EOLUS project has a very simple communication/computation pattern, which consists of cycles of computation followed by a reduce operation into a master node followed by a broadcast out of the master process. Although the amount of data involved in these reductions/broadcasts is very large (up to multi-Gigabyte), they are done relatively infrequently. Hence, the burst rate of these communications is not a concern. In fact, studies have shown that the communication/computation ratio in this code is about 5%. However, reduce and broadcast operations have the potential of being bandwidth bound because of contention for the master process. The easiest way to alleviate the congestion is to physically assign the master process to a network node that is over-designed in terms of network bandwidth. Given the fat tree topology of the network it would make sense for the master node to be the root node in the fat tree. Also note that reduce and broadcast will have hardware support in the Quadrics network; given a quality MPI implementation, these operations will perform well. The other way to alleviate this involves rewriting the application to implement a domain decomposition, which would eliminate the contention all together. However, the code developers consider this to be a complicated and lengthy software engineering task.

4.4 Studies Varying the SMP Size Using a Classified Application.

We also tried to address the question of how SMP size affects overall performance of these applications. To a first approximation, consider that inter-SMP MPI messages will incur a larger latency and a lower bandwidth. This would suggest a significant performance hit with smaller SMPs. However, the data we collected for a classified application on Blue Mountain do not show this.

We measured the total time to run the application in four different runs, each with 64 total processors but with 32, 16, 8, and 4 processors per SMP. We also kept the ratio of

HiPPI-800 links to processors constant at 1:4 for all the runs. Within a reasonable amount of error in the timings, the four runtimes are the same.

The reason that there is no SMP-size effect is that the application becomes self-synchronized as messages cross the inter-SMP boundary, regardless of how many inter-SMP boundaries there are. Even though the intra-SMP message passing is faster, the internal-SMP processors ultimately have to wait for messages from processors on the logical SMP boundaries, which in turn have to wait for messages from other SMPs. In addition, we maintained the surface-to-volume constant by preserving the ratio of inter-SMP links to processors constant across all runs.

***Conclusion*: There is no SMP-size effect on performance because the MPI-only applications becomes self-synchronized. The performance will be determined by the "slowest link in the chain".**

4.5 Studies Varying the Number of HiPPI Links Using a Classified Application.

Another way to probe the effect of available network bandwidth is to measure time for an application as a function the number of network interfaces available to it during the run. This is possible on the Blue Mountain system. Each 128-processor SMP on Blue Mountain is connected to the network via 12 HiPPI links. However, one can vary the number of HiPPI links available during a run using an environment variable. If the application is sensitive to network bandwidth its performance will continue to improve uniformly as more HiPPI links are enabled. (We have verified that the Blue Mountain MPI implementation does indeed scale message bandwidth linearly with the number of HiPPI links.)

Interestingly, although this experiment is carried out on Blue Mountain, the peak computation-to-communication ratio per SMP is similar to that of the Q system, adding to the validity of this approach. Additionally, although the topology of the complete Blue Mountain and Q interconnection networks are different (3-D Torus vs fat tree), this experiment was carried out only a portion of the Blue Mountain system, in which the interconnect per HiPPI link (i.e. rail) is a cross bar. Fat trees and crossbars have the same theoretical bisection bandwidth.

The data from this experiment using the classified application are shown as two curves in Figure 6. The top curve shows 8-box (1008-CPU) data and the bottom curve shows the same problem decomposed onto 16 boxes (2016 CPUs). The 8-box runs were done so that there was 12-way connectivity between all boxes and the 16-box runs were done so that there was 4-way connectivity between all boxes. (On Blue Mountain, the maximum connectivity between 16 boxes is 4-way.)

The data show that the application's performance responds to increasing bandwidth in the 100-400MB/s range but then reaches a plateau, in which further bandwidth in the network results in almost no performance gains. Had the application been sensitive to

network bandwidth its performance would have continued to improve uniformly as more HiPPI links were enabled.

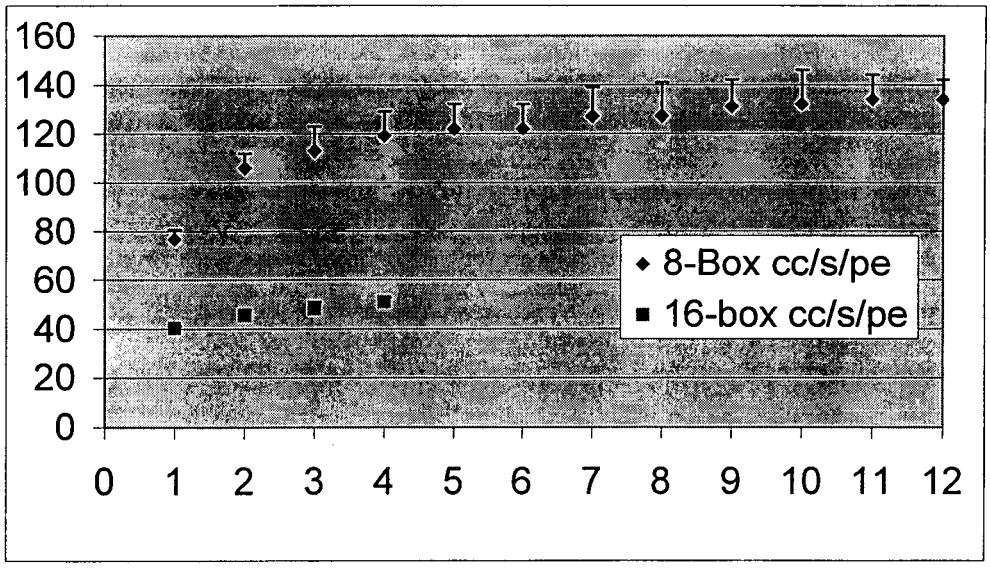

Figure 6. Performance of Classified App as a Function of Number of HiPPI links. Minimum and maximum cell-cycle-per-second-per-PE is plotted for two machine configurations.

***Conclusion*: Bandwidth is not a bottleneck for this app, as long as the network provides at least 400 MB/s.**

5. Reliability

Our comments assume the final system to be comprised of 2,992 ES45 machines, divided into 94 clusters.Let's assume that the ES45 hardware has a Mean Time Between Failure (MTBF) of 6,000 hours and ignore any downtime that may be introduced by software. With this MTBF, the probability that a given node will fail in 1 hour is:

Pfail(1) = 1/6000 = 0.00017,

so the probability of success of a given node during an hour is

Psuccess(1) = 1 - Pfail(1) = 0.99983.

For the system comprised of 2992 nodes, the probability of success for the entire machine is:

(Psuccess(1))**2992 = (0.99983)**2992 = 0.607, roughly 60% for one hour of uptime.

Looking at the probability of the entire cluster staying up for 2 hours, we have:

Pfail(2) = 2/6000 = 0.0003
Psuccess(2) = 1 - Pfail(2) = 0.9996 and
(Psuccess(2))**2992 = (0.9996)**2992 = 0.368

Note that a 32-node *ES40* cluster running AlphaServer SC Version 1 software has been reported (Roger Bartlett, LLNL) to take about 2 hours to boot, so there may be only a 37% chance that the entire machine would boot to completion before a node in the system would fail somewhere.
Psuccess(6) for this system is 5%.

By comparison, the 1-, 2- and 6-hour uptime probabilities for 374 GS320s (having 2000 hours MTBF) are respectively:
Psuccess(1) = 0.83
Psuccess(2) = 0.69
Psuccess(6) = 0.32
In order for the ES45 based-solution to exhibit comparable reliability, each node would need to have a MTBF of roughly 16,000 hours.

6. Operating System

Since ES45s are 4-processor nodes, we will be running 8 times more copies of the Tru64 kernel. Assuming that the OS runs on only 1 processor per node, 75% of an ES45 is available for computational use while 97% of a GS320 is available for computation. A 22% loss of processor availability translates into a loss of 6.5 Tflops that could otherwise have been used for number crunching.

Important note: this assumption of 75% proved to be too strong; recent Partisn runs by Randy Baker using all or 3 of the available processors in an SMP on the TC2K machine at Livermore showed some degradation (but not 25%).

7. System Administration

Roughly 3000 nodes would be nearly impossible to administrate by themselves, let alone in some cluster configuration. The scalability of the TruCluster management tools is already under scrutiny because it is not performing satisfactorily for even 16 nodes.

8. Network

8.1 Static Allocation of Rails

PCI buses on the ES-40 exhibit a bandwidth of approximately **200 Mbytes/s for unidirectional traffic.** However, in the presence of **bidirectional traffic**, the effective unidirectional bandwidth drops to somewhere around **90 Mbytes/s**. In order to alleviate this potential bottleneck, Compaq proposed a "static" allocation of rails, based on the following constraints:

a. each rail interface is dedicated to either incoming or outgoing traffic;
b. there must be full connectivity between SMPs, i.e. each SMP needs to be connected to any other SMP in both directions;
c. there must be direct connectivity, i.e. no message between any two SMPs can be routed via another SMP;
d. rails must be independent: messages cannot pass from one rail to another

We have derived (and proven) an algorithm that gives the lower bound for the number of rails needed to satisfy these constraints as a function of number of SMPs in the network; Figure 7 shows a plot for a range up to 4,096 SMPs. Note that the minimum number of rails required for a 4-processors-per-SMP system (~12000 processors total) is 14. Of course, this many rails cannot be accommodated by the existing ES-40 hardware in any proposed configuration. Thus, all/some of the constraints need to be relaxed. In all likelihood, a dynamic allocation of the rails will be needed; however, as noted above, this will result in decreased bandwidth in/out of each SMP.

***Conclusion:* Static rail allocation is not possible. A dynamic allocation is needed, but we will pay the price of decreased bandwidth for bidirectional traffic.**

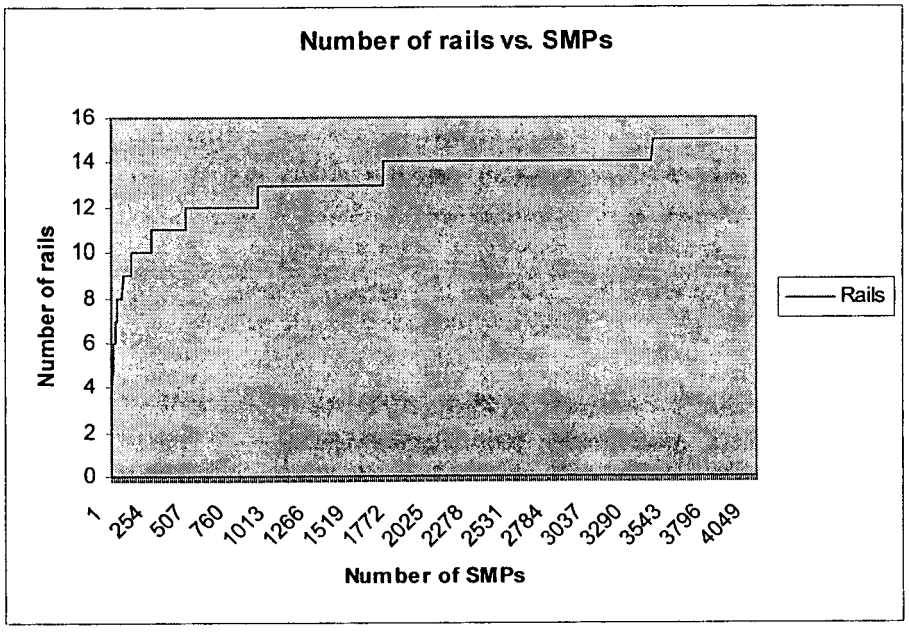

Figure 7. Number of rails needed for the static allocation.

8.2 Striping

One possible means for improving the latency-bandwidth balance is message striping across rails. This means that instead of having MPI use a single rail for all the outgoing/incoming messages from a given process, implement MPI so that it divides up the messages and stripes the segments through multiple rails concurrently. Intuitively, by using this mechanism we improve bandwidth but pay an increased latency cost. However, our analysis shows that it will be very difficult to implement this feature in MPI.

The reason lies in current limitations of the PCI buses in the ES40/Wildfire machines, *to the best of our knowledge and based on discussions with the network designers at Quadrics.* In order to implement multi-rail striping in MPI, multiple Elan NICs in a single SMP would need to communicate with one another (to implement MPI send/receive protocols). However, direct communication between PCI buses is not possible, making communication between Elans extremely inefficient. In the current architecture, communication between Elans can be done only through the main SMP memory, involving at least two PCI bus transmissions. Moreover, the Elan would need to poll common shared memory locations in main memory, which would be a very inefficient/costly mechanism.

***Conclusion:* The feasibility of striping in MPI on the current architecture (Quadrics/ES/40Wildfires) is highly questionable. Striping is a bandwidth-enabling solution for the case of some global communication patterns such as broadcast and reduce and for the case of bandwidth intensive I/O.**

8.3 Topology

A network with ~4000 nodes would require a height-6 to height-10 fat tree (depending on implementation) to connect them. A reduced number of rails per SMP (e.g., 2 *vs.* 8) would introduce an increased probability of failure in the network. Also, as a result of the vastly larger number of switches in the network the probability of failure increases manifold.

9. Conclusions

We have shown that major apps from ASCI workload, **Rage, Partisn and a classified application, are not bandwidth sensitive and that, given the overall design of the network, the decrease in bandwidth from 500 to 200 Mbytes/s will not lead to any significant performance degradation**. Moreover, Rage would have to communicate roughly two orders of magnitude more data than it does now to achieve bandwidth saturation. This is an unlikely possibility due to the necessary increase in memory sizes, of roughly three orders of magnitude for Rage, that would be needed. In addition, our analysis indicates that the Q network is adequate at sustaining heavy traffic.

The EOLUS project app has a different communication pattern than the other apps analysed, and while in principle, this pattern can become bandwidth bound, we identified ways to alleviate this.

The report presented a qualitative and quantitative analysis. Our conclusions are based on input from actual runs on Blue Mountain and Nirvana, accurate simulation of the Q network and on models validated on all ASCI platforms.

We also touched on the issues of reliability and operating system overheads. We signal a potentially significant extra overhead from reducing the number of processors per SMP due to the fact that, since one processor per SMP handles system tasks, more processors will be dedicated to running OS.

We indicated a concern related to the increased complexity of system administration activities.

We have shown that the static allocation of rails proposed by Compaq is not achievable. In fact, this approach would require a impractical number of rails. A dynamic allocation of rails will be needed, but it will come at a steep price in terms of available bandwidth for bidirectional traffic, as that encountered in some of our important applications. We also analyze striping of large messages across rails, coming to the conclusion that due to a limitation in the design of the PCI bus this bandwidth-enabling solution will not be feasible in MPI.